International Journal of Control and Automation
Vol. 8, No. 6 (2015), pp. 339-348
http://dx.doi.org/10.14257/ijca.2015.8.6.33

Study on the NAND Flash Memory-Based Failure Recovery Improvement Using T*-Tree

Seong-Soo Han[1*] and Sung-Je Cho[2]
[1*]*Ed.D.,Bucheon University, Bucheon, Korea*
[2]*Department of Education,Dongbang Culture Graduate University, Seoul, Korea*
[1]*postsky0@naver.com*, [2]*chosj715@daum.net*

Abstract

As compared to other types of flash memory, NAND flash memory is attracting attention as the next generation storage device for its low power use and fast access speed, which make it suitable for ubiquitous and mobile environment. In particular, flash memory is being used in storage devices in many fields, including the memory system of small mobile devices, as well as SSD (solid state disk) of large memory. Active research is underway for the efficient use of flash memory. However, NAND flash memory based storage devices exhibit unique hardware characteristics, such as erasing structure, and the B-tree index structure causes performance degradation when repeated writing requests are made. In order to enhance these problems, the present paper proposes to use NAND flash memory-based failure recovery improvement technique using the T-tree. After conducting performance evaluation of the proposed technique and the existing technique, the proposed technique was found to be more efficient in its performance by 69%, compared to the existing technique.*

Keywords: *B-Tree, T*-Tree, NAND Flash Memory, Recovery*

1. Introduction

Recently, NAND flash memory is gaining attention as the next generation storage device for its characteristics of being non-volatile, portable, low power, and highly durable. With the increase in the use as mobile computing storage medium and the fact that the high capacity SSD (NAND Flash-Based Solid State Disk) uses the same interface as that of hard disk drive, it is expected to replace hard disk drives in many systems that require high capacity storage devices [1, 2, 10].

However, the 'Erase-Before Write' structure of NAND flash memory is asymmetrical in its speed and processing unit for reading and writing, hence directly using the application that was used in hard disk can result in many shortcomings. In order to address this problem, the FTL (Flash Translation Layer) [2] was proposed. The FTL removed the mentioned shortcoming by regarding logical address as physical address.

Still, using application that generates a frequent position update causes significant performance degradation in storage devices. The generation of B-tree for managing high capacity data, especially, frequently requests the position update in the same area. To address this problem, the technique was proposed to use T*-tree, instead of B-tree, and to achieve higher performance in failure recovery [9].

In the present study, the T*-tree index was used to build the log directory and the T*-ISLD technique ((T*-Tree Index Segment Log Directory technique, or hereinafter referred to as the T*-ISLD technique) was used to build the recovery system in case of failure. The proposed technique from this study manages the changed information in the log to assist failure recovery. It also commits all data when changing the root node and performs checkpoint. Then the recovery is performed by using the log information recorded in the event of failure. In conclusion, the proposed technique enables building of safe, high performance T*-tree index by using a buffer in the NAND flash memory.

The present paper is organized as follows: related researches are presented in Chapter 2, followed by the explanation about the proposed technique, performance comparison evaluation, and conclusion and future research direction in Chapter 3, Chapter 4, and Chapter 5, respectively.

2. Related Researches

2.1. NAND Flash Memory Method Using the B+-Tree

In a system structure that uses NAND flash memory-based SSD, the B+-tree is used to compose SSD and NAND flash memory chip is made up of a set number of blocks. Each block is divided into 4096-byte Data Area, for data storage, ECC (Error Correction Code) for error correction, and 64-byte Spare Area for meta information storage. In the NAND flash memory, three basic operations, reading, writing, and erasing, are provided.

In the NAND flash memory, data retrieval is performed through data structure and algorithm using the B+-tree. However, a B+-tree node is composed of multiple entries and inserted by an entry when stored. This causes the occurrence of repeated writing in certain areas, and the repeated writing concentrated in one specific area leads to a significant performance degradation due to the characteristic, in which the position update is not allowed in flash memory. The intensive writing operation generated in a small area during building the B+-tree lowers the building performance in the NAND flash memory [3].

2.2. NAND Flash MR-Tree Delay Computation Method

MR-tree [4] is a variation of R-tree and a spatial index structure with access to spatial data. Since it is important to reduce cache miss when the difference between the CPU speed and the memory speed increases in the main memory database system, the MR-tree enhances the utilization rate of the middle node entry compared to the R-tree and reduces the height of the tree to respond sensitively to the movement of cache.

In addition, applying the indexing technique to flash memory, it is possible to approach and manage high capacity data efficiently. However, the tree-index is prone to insert, delete, split, and other modifications. Furthermore, the modification of even one node entry can cause performance degradation due to the occurrence of writing and erasing operations. In order to address this problem, realization of efficient R-tree in the BFTL and flash memory system reduced the number of writing operation of the buffering layer to improve the performance degradation, but it created additional retrieval operation [4].

2.3. NAND Flash File Crash Recovery Techniques

This technique is used to efficiently initialize when the file system crashes and terminates abnormally. By setting the working area, it allows writing operation in a certain area of the entire flash memory space at a specific point in time. Then the data in the operation area is saved in the designated space, and they are designed to determine the most recently set up working area during the initialization. Under this structure, file system can be initialized because only the most recent working area is being retrieved for the consistency in the event of crash. Therefore, the time required to recover crash increases proportionally to the size of the working area. This technique can further assist in solving the scalability problem since the crash recovery time increases proportionally to the memory. However, the technique does not suggest that the crash time is proportional to the flash memory capacity [5].

2.4. IBSF Technique

In this technique, the IBSF refers to the software module that can be inserted on top of the flash conversion layer and file system. In addition, it was proposed in order to build effective B-tree in the flash memory. ISBF generates index unit with the changing information during the B-tree establishment, and temporarily retains information in the index buffer. When the index buffer is filled, IBSF updates the node by collecting the first index units that are related to the first index unit. Furthermore, the IBSF retrieves the repetitive index unit in case of unit buffer insertion. Through this process, IBSF delays the moment, in which the buffer is full and reduced the number of writing operation, hence the cost was reduced.

However, since the index buffer is volatile memory space, it is exposed to risk of loss during sudden power disconnection or power failure. If the index data is lost, the IBSF does not have a way of identifying the lost data, therefore it is impossible to recover but since the consistency of the tree is destroyed, an additional way for the recovery is needed [6].

2.5. BISLD Technique

This technique (B+-Tree Index Segment Log Directory technique: hereinafter referred to as the BISLD technique) is a failure recovery technique that uses the B+-tree technique and directory log buffer [7]. Among various logging techniques, it applies the delayed technique for the recovery. After the completion of work, it proceeds with recovery through re-operation during recovery and with the updated operations in the log directory. The insert algorithm in the BISLD technique is as follows. ① Retrieve the leaf node to be inserted. ② Retrieve the node with smaller value than the key value of the index node. ③ If the key to be inserted is greater than the key value of the index node, follow the node with the greater value. ④ Retrieve the next node. ⑤ Repeat until the leaf node for the insertion is found [7].

This technique uses the B+-tree and is susceptible to frequent position update request in the same area during the work. For that reason, overhead occurs during failure recovery.

2.6. T*-Tree Index Technique

This technique uses the T*-tree, in which all data are stored in the main memory device as opposed to the index techniques that are based on the existing disk. This index structure modifies the disadvantages and allows faster access to data and efficient use of the memory, which are the advantages of the T-tree index structure used in the main memory database systems [8].

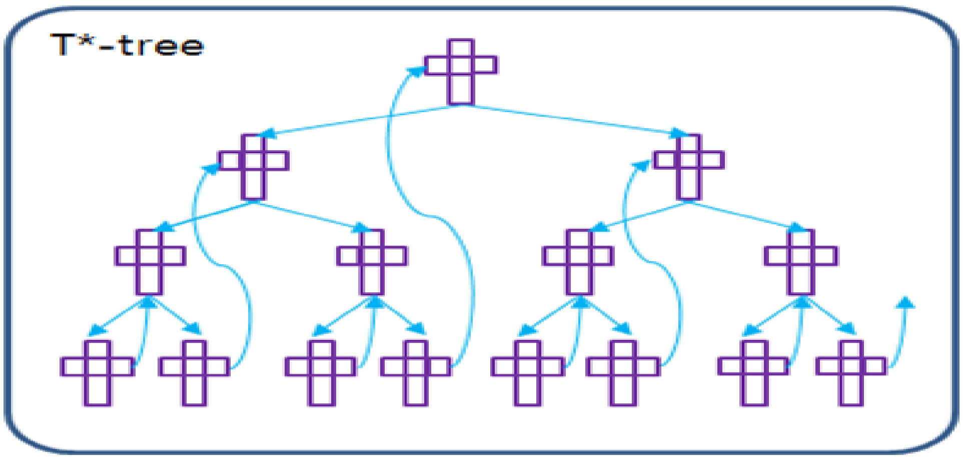

Figure 1. T*-Tree

The structure of the T*-tree is as shown in [Figure 1]. It consists of additional rear pointer to the T-node and commands the rear nodes consisted of succeeding size of nodes to achieve the configuration of simple connection list according to the size of the nodes. In addition, the inserted data item within the nodes are stored in order from the left, unlike in the T-tree, so that the empty space is always to the left. Each element in the node is configured with single key value and the address of the data item with the key value, and they are pre-sorted according to the size of the item [7].

As shown in [Figure 1], the added rear pointer to the T*-tree points at the rear nodes configured of succeeding values. Therefore, the in-order tree traversal of the T-tree is improved and the direct traversal to the next is made possible, reducing the length of the traversal path. In addition, when the nodes are generated, the first item, of the maximum value, is stored from the right in an order. This allows the maximum overflowed value to be stored at the right-most side of the empty space of the LUB node in case of overflow due to the data insertion. In case of underflow due to the deletion, the minimum value from the LUB node, or the left-most item, gets borrowed. In order to prevent from generating additional data alignment or movement within the nodes and directly access the LUB node using the rear pointer without going through the middle nodes, the T*-tree structure and processing algorithm are improved. In addition, for the efficient use of the storage space, additional rear pointer has been added to the T-node structure for the T*-tree, when compared to the existing T-tree, while also having the same internal structure. Since the item per pointer ratio within the node does not differ greatly, it still retains the advantage of the T-tree as having efficient storage space [8].

3. NAND Flash Memory-Based Failure Recovery Method Using T*-Tree

3.1. T*-ISLD Method Recovery System Structure

The recovery structure of the T*-ISLD consists of the T* index and log directory and the configuration is as shown in [Figure 2]. The T*-tree algorithm is applied in insertion and retrieval [8].

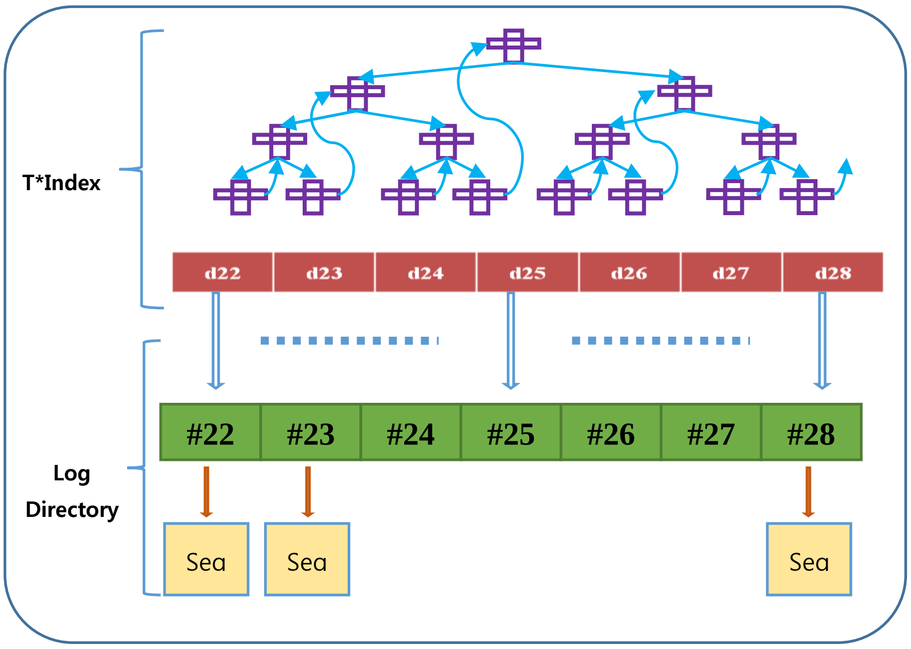

Figure 2. Recovery Structure in T*-ISLD Technique

Applying the algorithm, recovery is performed promptly by rapid retrieval of RedoLog, when failure occurs to a specific flash memory. This is because the T*-tree is superior during insertion and retrieval, compared to the other indices [8].

The present technique creates the index unit when the T*-tree is changed, maintains it in the buffer, and performs the commit when the buffer is full. It also records and keeps the changed information in the log, commits all index units when the root node is changed, and performs checkpoint.

3.2. Commit Policy

The commit policy proposed in the present paper is as follows. When the buffer is full, the index units related to the first index unit should be gathered to be committed to one node. When the root node is changed, all index units within the buffer shall be committed and checkpoint shall be performed.

3.3. Failure Recovery Algorithm

The failure recovery algorithm structure of the proposed technique is as shown in Figure 3. Insertion and retrieval are applied from the T*-tree algorithm. This algorithm promptly retrieves RedoLog in case of a failure in a specific flash memory, and enables a quick recovery. This is because the T*-tree index structure is designed to minimize the memory use and dynamically adapt, so that the insertion and retrieval of data are superior to that of the other indices [8].

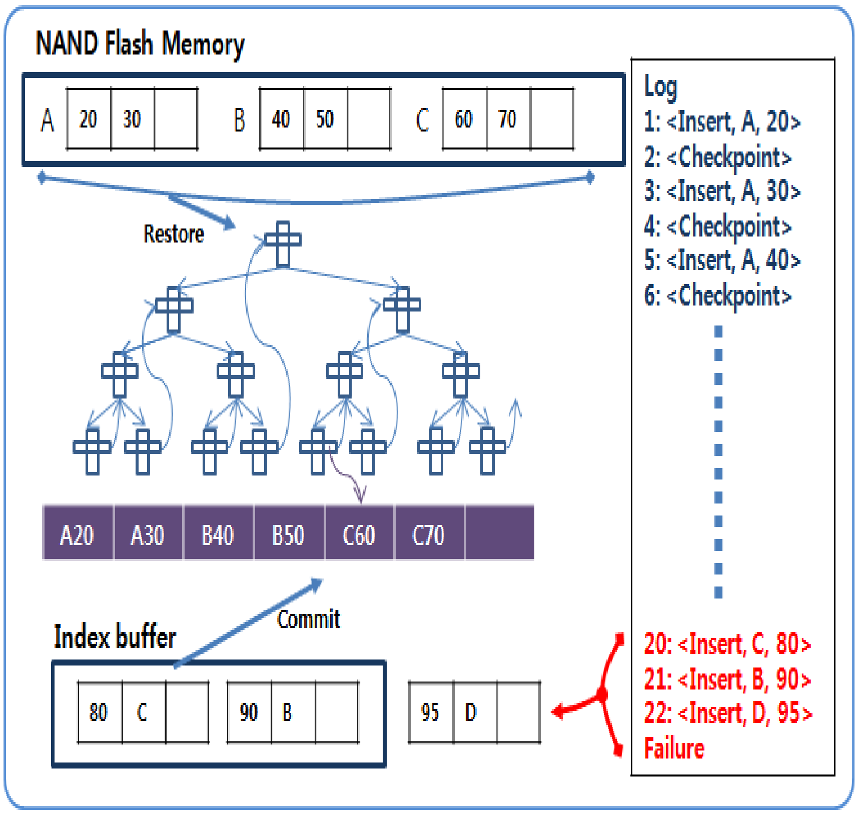

Figure 3. Failure Recovery Algorithm

The log record of the failure recovery can be divided into Redo and Undo commands. In the present study, the log record is managed in the log and has the form of <command, node number, key number> during the changes in the T*-tree. For example, <Insert, A, 20> is a command to insert a key value of 20 to the A node. The proposed technique performs checkpoint, records the information in the log, and recovers by performing the command after the last checkpoint during the failure recovery. The recovery order here is to first use the stored T*-tree node to build the T*-tree, and then recalculate the index unit using log record information to insert to the next index buffer. In addition, for the failure recovery, the stored A, B, and C noes in the flash memory are used to build the T*-tree, and then the log information is used to perform recovery to before the failure.

4. Performance Evaluation

In the present study, performance evaluation was conducted to compare the performances of the BISLD technique, which uses the existing B+-tree, and the proposed T*-ISLD technique, which uses the T*-tree. The system environment for the performance evaluation was Intel CPU Core i7-4790(3.49GHz), 84GB RAM, OS Windows 7. The operation form of the performance evaluation measured the amount of time required to perform retrieval of the specific log. The simulation was done to compare performances of two techniques by varying the number of index tree and investigating the change in time when retrieving log information for recovery.

4.1. Performance Comparison Analysis

The performances of recovery techniques using the proposed T*-ISLD technique and existing BISLD technique were compared and analyzed. The results are as shown in Table 1.

The number of input index tree varied from 15,000 to 60,000, in increments of 5,000. The recovery processing time (unit: ms) varied from 0 to 17, showing the distribution from the minimum of 2.82ms when using the T*-ISLD technique to the maximum of 13.87ms when using the BISLD technique. The improvement ratio, which compares and analyzes performances of two techniques, is also indicated in Table 1.

Table. 1 Comparison Table of the Recovery Performance

Quantity Index Tree	BISLD Techniques	T*-ISLD Techniques	Improvement Ratio
15,000	13.87	9.40	32%
25,000	14.61	6.05	59%
35,000	15.10	4.52	70%
40,000	15.29	4.02	74%
45,000	15.46	3.62	77%
50,000	15.61	3.30	79%
55,000	15.75	3.04	81%
60,000	15.87	2.82	82%

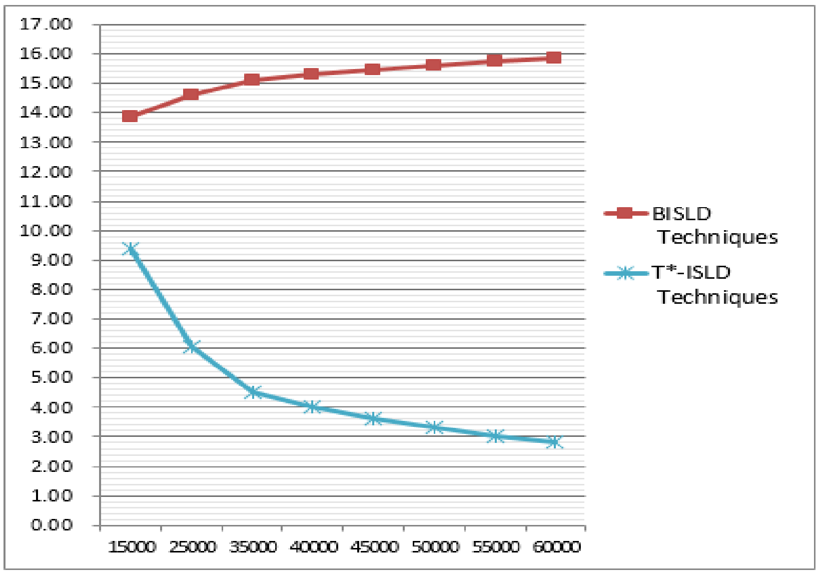

Figure 4. Performance Comparison of Recovery Time

Figure 4 shows the results for the BISLD technique and the proposed technique. As shown in the graph, when the number of index tree was 15,000, the result of the T*-ISLD technique was 9.40 and that of the BISLD technique was 13.87, indicating 32% improvement for the T*-ISLD technique. When the index tree was 35,000, the results of the T*ISLD technique and the BISLD technique were 4.52 and 15.107, respectively, indicating 70% improvement. When the index trees were 50,000 and

60,000, the results of the T*-ISLD technique was 3.30 and 2.82, respectively, and that of the BISLD technique were 15.61 and 15.82, respectively, indicating the improvement ratio of 79% and 82%, respectively. On average, T*-ISLD technique showed 69% improvement in performance compared to the BISLD technique.

5. Conclusion

The proposed technique of the present study reduced cost of failure recovery by using buffer to build high performance T*-tree in NAND flash memory, by maintaining and managing changed information upon node change of the T*-tree, performing checkpoint when updating root node, and building T*-ISLD using T*-tree in NAND flash memory.

Lastly, through experiment, the performances of the BISLD technique, which uses the existing B+-tree, and that of the T*-ISLD technique, which uses the proposed T*-tree, were compared. It was found that the performance of the proposed technique was superior.

The results of performance evaluation for the existing BISLD technique and the T*-ISLD technique are as follows. The number of index tree for retrieval was varied from 15,000 to 60,000 for the performance evaluation. When the number of index tree was 15,000, the result of the T*-ISLD technique was 9.40 and that of the BISLD technique was 13.87, indicating 32% improvement for the T*-ISLD technique. When the index tree was 35,000, the results of the T*ISLD technique and the BISLD technique were 4.52 and 15.107, respectively, indicating 70% improvement. When the index trees were 50,000 and 60,000, the results of the T*-ISLD technique was 3.30 and 2.82, respectively, and that of the BISLD technique were 15.61 and 15.82, respectively, indicating the improvement ratio of 79% and 82%, respectively.

As a result, the T*-ISLD technique showed an average of 69% improvement in its performance, compared to the BISLD technique, and as the number of index tree increased, the performance of the T*-ISLD technique improved more than that of the BISLD technique. The reason accounts for the fact that T*-tree has superior performance than the B+-tree, due to the difference in the index structure, in which the T*-tree index structure resulted in superior performance than the others.

In the future, ways to reduce overhead for the failure recovery in NAND flash memory will be studied, as well as the failure recovery techniques of flash memory that can be used in various fields.

References

[1] S. E. Corporation, "PM810," Data Sheet, **(2011)**.

[2] T.-S. Chung, “A survey of Flash Translation Layer”, Journal of Systems Architecture, vol. 55, **(2009)**, pp. 332-343

[3] Samsung Electronics Company, K9PDG08U5D 128G bit*8 bit NAND Flash-Memory Data Sheet, **(2004)**

[4] K.-C. Kim and S.-W. Yun, “MR-Tree: A cashe-conscious main memory spatial index structure for mobile GIS”, Web and wireless geo-graphic information systems, The 4th international workshop, **(2004)**, pp. 167-180.

[5] H. Park and C. Yoo, “A Design of Efficient Crash Recovery Technique for NAND Flash File System”, Korea Information Science Society Journal, vol. 35, no. 2(B), **(2008)**.

[6] H.-S. Lee, B.-K. Kim, Y.-D. Joo and D.-H. Lee, “An Efficient Recovery Management Scheme for an Index Buffer of B+tree Based on NAND Flash Memory”, Korea Information Science Society, Database Research, vol. 27, no. 3, **(2011)**, pp. 21-34.

[7] S.-J. Cho and S.-S. Han, “Recovery Model Improvement Using BISLD in Mobile Computing Environment”, The Journal of Korea Academia-Industrial Cooperation Society, vol. 13, no. 10, **(2012)**, pp. 4786-4793.

[8] K.-R. Choi, K.-R. Kim and K.-C. Kim, "T*-tree: An Efficient Indexing Technique for Main Memory Database", Journal of The Korean Institute of Communications and Information Sciences, vol. 21, no. 10, **(1996)**, pp. 2601

[9] S.-S. Han and S.-J. Cho, "A Study on the Improvement of Failure Recovery Based on a AND Flash Memory Environment", Proceedings of the 6th International Workshop Series, Jeju National university International Center, Jeju Island, Korea, **(2015)** April 15-18.

[10] Y. Ryu, "A Buffer Management Scheme for mobile Computers with Hybrid Main Memory and Flash Memory Storages", The Journal of IJMUE, vol. 7, no. 2, **(2012)**, pp. 235-240.

Authors

Seong-Soo Han

• February 2000: Gyeongsang National University Gyeongsang National University Telecommunication Engineering (Bachelor of Engineering)

• August 2005: Soonchunhyang Industrial Information Graduate School of telecommunications (Master of Engineering)

• February 2014: Dongbang Graduate School of Education (Doctor of Pedagogy /IT convergence)

<Interest Areas> DBMS and big data, IT convergence, mobile computing, Flash Memory

Sung-Je Cho

• February 1977: Hongik University Electrical Calculation Major (Doctor of Science)

• March 2007: Professor of Department of Education at Dongbang Graduate School

<Interest Areas> DBMS and big data, IT convergence, mobile computing